Esercizi alberi

Esercizio

- Sia data la seguente struttura per la memorizzazione di alberi binari etichettati con numeri interi:

```
typedef struct nodo {
        int info;
        struct nodo *left, *right;
} NODO;
typedef NODO *tree;
```

Si devono scrivere due funzioni ricorsive

```
int sommaNodi(tree t);
int cercaMax(tree t);
```

delle quali, sommaNodi somma i valori delle etichette nell'albero, mentre cercaMax cerca il valore dell'etichetta massima dell'albero.

```
int sommaNodi(tree t) {
    if (t==NULL)
        return 0;
    return t->info+sommaNodi(t->left)+sommaNodi(t->right);
}

int sommaNodi(tree t) {
    int s=0;
    if (t==NULL)
        return 0;
    s=t->info;
     if(t->left!=NULL)
        s += sommaNodi(t->left);
     if(t->right!=NULL)
        s += sommaNodi(t->right);
    return s;
}
```

```
int max(int a,int b) {
    if(a>b)
        return a;
    else
        return b;
}

int max3(int a,int b,int c) {
    return max(a,max(b,c));
}

int cercaMax(tree t) {
    int s,d,m;
    if (t==NULL)
        return 0;//eseguita solo se da subito l'albero è vuoto
    if (t->left==NULL && t->right==NULL)
        return t->info;
    if(t->left==NULL)
        return max(t->info, cercaMax(t->right));
    if(t->right==NULL)
        return max(t->info, cercaMax(t->left));
    return max3(cercaMax(t->right), cercaMax(t->left), t->info);
}
```

Esercizio (tde 26-1-2009)

- Si consideri la seguente definizione di un albero binario (binario=con due rami in ogni nodo):

  typedef struct EL { int dato;

  struct EL * left, * right; } node;

  typedef node * tree;

- Codificare una funzione che riceve in input due alberi e restituisce 1 se i due alberi sono identici, 0 altrimenti.

```
int f(tree t1,tree t2) {
   if(t1==NULL && t2==NULL)
        return 1;
   if(t1==NULL || t2==NULL)
        return 0;
   return (t1->dato==t2->dato &&
              f(t1->right,t2->right) && f(t1->left,t2->left))
}
```

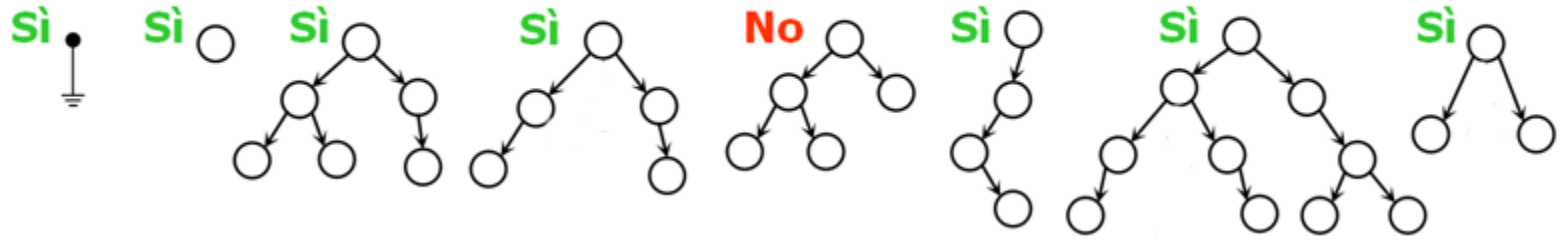
Esercizio

- Un albero binario si dice isobato se tutti i cammini dalla radice alle foglie hanno la stessa lunghezza
- Data la seguente definizione di albero

```
typedef struct EL    {      int dato;
                            struct EL *left;
                            struct EL *right;   } Node;

typedef Node *tree;
```

codificare una funzione che riceve in input un albero e restituisce 1 se l'albero è isobato, 0 altrimenti.

Attenzione: si considerino attentamente gli esempi sottostanti.



Uso funzione ausiliaria

```
int contaProfonditaSeIsobato(tree t) {
    int s, d;
    if(t==null)
        return 0;

    s=contaProfonditaSeUguale(t->left);
    d=contaProfonditaSeUguale(t->right);

    if(d==-1||s==-1)
        return -1
    if(d==s)
        return d+1;
    if(d==0)
        return s+1;
    if(s==0)
        return d+1;

    return -1;//d!=s
}
```

```
int isobato(tree t) {
    if(contaProfonditaSeIsobato(t)==-1)
        return 0;
    else
        return 1;
}
```

```
int maxdepth ( tree t ) {
  int D, S;
  if (t == NULL)
    return 0;
  S = maxdepth( t->left );
  D = maxdepth( t->right );
  if ( S > D )
     return S + 1;
  else
     return D + 1;
}
```

```
int mindepth ( tree t ) {
  int D, S;
  if (t == NULL)
    return 0;
  S = mindepth( t->left );
  D = mindepth( t->right );
  if ( S==0 )
     return D + 1;
  if ( D==0 )
     return S + 1;
  if ( S < D )
     return S + 1;
  else
     return D + 1;
}
```

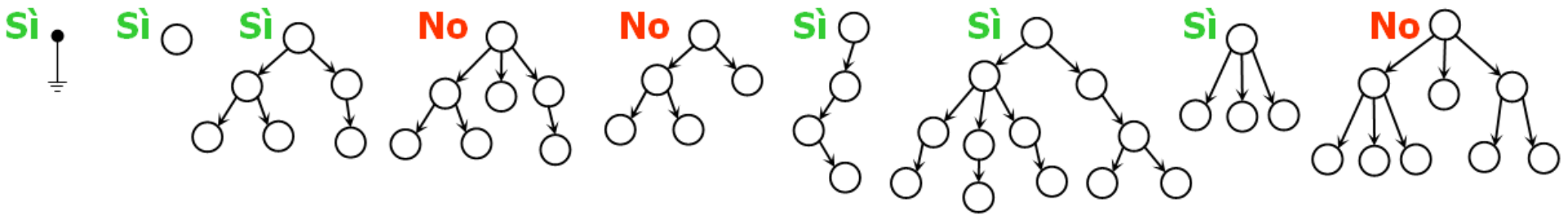
```
int isobato ( tree t ) {
        return maxdepth(t)==mindepth(t);
}
```

Esercizio

- Si consideri un albero ternario privo di dati e rappresentato dalla seguente definizione ricorsiva:

```
typedef struct Elemento {
        int dato;
        struct Elemento * left, * center, * right; } Nodo;
typedef Nodo * Tree;
```

- Si progetti un algoritmo ed eventualmente si codifichi una funzione C che restituisce 1 se **tutti i cammini** dalla radice dell'albero alle foglie hanno la **stessa lunghezza**, e restituisce 0 altrimenti.



```
int isobatoAux( Tree t, int * depth ) {
  int dl, dc, dr, il, ic, ir;    /* 3 profondità per i rami e 3 booleani */
  if ( t == NULL ) {
    *depth = 0;                  /* Un albero vuoto è isobato di profondità 0 */
    return 1;
  }
  il = isobatoAux(t->left,   &dl);  /* Controlliamo l'isobaticità dei rami    */
  ic = isobatoAux(t->center, &dc);  /* Memorizzando le (eventuali) profondità */
  ir = isobatoAux(t->right,  &dr);
  if ( !il || !ic || !ir || dl != dc || dc != dr )
    return 0;

  *depth = dl + 1;  /* tanto le profondità sono tutte uguali */

  return 1;
}
```

```
int isobato ( Tree t ) {
  int foo=0;
  return isobatoAux( t, &foo );
}
```

Esercizio

- Un albero N-ario generico è un albero i cui nodi possono avere un numero arbitrariamente grande di rami uscenti. Si definisca una struttura dati adatta a rappresentare un albero N-ario. Per semplicità si consideri il caso in cui i nodi contengono, come dati utili, dei semplici numeri interi.
- Ogni nodo contiene, invece di una coppia di puntatori a nodi, come nel caso degli alberi binari, una lista di puntatori a nodo. Tale lista è una lista concatenata semplice, realizzata tramite la struttura **Ramo**.

  struct Knot;

  typedef Knot * Albero;

  typedef struct Branch { Albero child; struct Branch * next; } Ramo;

  typedef struct Knot { int dato; Ramo * rami; } Nodo;
- Si progetti (e/o codifichi) la funzione **int conta(** … **)** che conta il numero di nodi di un albero N-ario.

Si può utilizzare una funzione ricorsiva che per ogni nodo valido restituisce 1 più il totale dei nodi accumulato scandendo la lista dei puntatori ai sottoalberi, tramite un ciclo while.

```
int conta ( Albero t ) {
  int n; Ramo * cur;
  if ( t == NULL ) {
    return 0;
  } else {
    n = 1;
    cur = t->rami;
    while ( cur != NULL ) {
      n += conta( cur->child );
      cur = cur->next;
    }
    return n;
  }
}
```

Se ne può anche scrivere una versione più elegante, completamente ricorsiva, che non usa il ciclo while per scandire la lista dei sottoalberi.

```
int contaNodi ( Albero t ) {
  if ( t == NULL )
    return 0;
  else
    return 1 + contaRami( t->rami );
}

int contaRami ( Ramo * b ) {
  if ( b == NULL )
    return 0;
  else
    return contaNodi( b->child ) + contaRami( b->next );
}
```

Esercizio

- Si consideri la seguente definizione di un albero binario (binario=con due rami in ogni nodo):

  typedef struct EL { int dato;

  struct EL * left, right; } node;

  typedef node * tree;
- Definiamo un albero come "artussiano" se è composto solo da
  - nodi foglia
  - nodi con un solo figlio
  - nodi con due figli aventi lo stesso numero di discendenti
- Codificare una funzione che riceve in input un albero e restituisce 1 se l'albero è "artussiano", 0 altrimenti. Nel risolvere il problema è consigliato servirsi di opportune funzioni ausiliarie.

```
int contaNodi ( tree t ) {
  if ( t == NULL )
    return 0;
  else
    return (contaNodi(t->left) +
                contaNodi(t->right)
                 + 1); /* c'è anche il nodo corrente */
}
```

```
int artussiano(tree t) {
    if(t==NULL)
        return 1;
    if(t->left==NULL && t->right==NULL)
        return 1;
    if(t->left==NULL)
        return artussiano(t->right)
    if(t->right==NULL)
        return artussiano(t->left)
    if(contaNodi(t->left)==contaNodi(t->right) &&
                    artussiano(t->left) && artussiano(t->right))
        return 1;
    return 0;
}
```

Esercizio

- Si consideri la seguente definizione di un albero binario (binario=con due rami in ogni nodo):

  typedef struct EL { int dato;

  struct EL * left, right; } node;

  typedef node * tree;

  in cui dato assume sempre valori positivi.
- Supponiamo che percorrendo un cammino dalla radice alle foglie si totalizzi un punteggio pari alla somma dei valori contenuti nei nodi percorsi.
- Scrivere una funzione maxPunti che calcola il punteggio massimo che possiamo totalizzare percorrendo un cammino dalla radice alle foglie.
- Vogliamo percorrere l'albero dalla radice ad una foglia totalizzando esattamente un certo punteggio: né far meno, né sforare. Scrivere una funzione esisteCammino che dati un albero ed un intero k, dice se esiste un cammino dalla radice ad una foglia che permette di totalizzare esattamente k punti.

```
int maxPunti ( tree t ) {
  int D, S;
  if (t == NULL)
    return 0;
  S = maxPunti( t->left );
  D = maxPunti( t->right );
  if ( S > D )
      return S + t->dato;
  else
      return D + t->dato;
}
```

```
int esisteCammino ( tree t, int k ) {
  int D, S;
  if (t == NULL && k==0)
      return 1;
  if (t == NULL)
      return 0;
   if (k – t->dato <0)
      return 0;
 if( t->left==NULL)
          return esisteCammino(t->right, k - t->dato);
if( t->right==NULL)
          return esisteCammino(t->left, k - t->dato);
return (esisteCammino(t->left, k - t->dato)  ||
            esisteCammino(t->right, k - t->dato));
}
```

Esercizio

- Si consideri la seguente definizione di un albero binario (binario=con due rami in ogni nodo):

  typedef struct EL { int dato;

  struct EL * left, right; } node;

  typedef node * tree;
- Scrivere una funzione che riceve il puntatore alla radice di un albero e lo scandisce interamente costruendo una lista tale che abbia tanti nodi quanti sono i nodi dell'albero e che ogni nodo della lista punti ad una diversa foglia dell'albero. La funzione deve restituire al chiamante il puntatore all'inizio della lista creata.

  typedef struct ELLista { tree foglia;

  struct ELLista * next; } nodeLista;

  typedef nodeLista * Lista;
- Si noti che esistono diversi modi di visitare l'albero che originano diverse liste come risultato.

```
Lista demiurgo ( tree t ) {
   return demiurgino(t,NULL)
}

Lista demiurghino ( tree t, Lista L ) {
     if(t==NULL)
        return NULL;

     L=insInCoda(L,t);

     if(t->left==NULL && t->right==NULL)
        return L;
     if(t->left==NULL)

        return demiurghino(t->right,L);
     if(t->right==NULL)
        return demiurghino(t->left,L);

     L=demiurghino(t->left,L);
     L=demiurghino(t->right,L);

     return L;
}
```

Esercizio

- Si consideri la seguente definizione di un albero binario (binario=con due rami in ogni nodo):

  typedef struct EL { int dato;

  struct EL * left, right; } node;

  typedef node * tree;
- Scrivere una funzione che riceve il puntatore alla radice di un albero e lo scandisce interamente costruendo una lista tale che abbia tanti nodi quante sono le foglie dell'albero e che ogni nodo della lista punti ad una diversa foglia dell'albero. La funzione deve restituire al chiamante il puntatore all'inizio della lista creata.

  typedef struct ELLista { tree foglia;

  struct ELLista * next; } nodeLista;

  typedef nodeLista * Lista;
- Si noti che esistono diversi modi di visitare l'albero che originano diverse liste come risultato.

```
Lista creaLista(tree t, Lista lis) {
    if (t==NULL)
        return NULL;
    if(t->left==NULL && t->right==NULL)
        return inserisciInCoda(t,lis);
    if(t->left==NULL)
        return creaLista(t->right,lis);
    if(t->right==NULL)
        return creaLista(t->left,lis);
    return creaLista(t->left, creaLista(t->right,lis));
```

Esercizio

- Si consideri la seguente definizione di un albero binario:

```
typedef struct Elemento { char parola[30];
                          int occorrenze;
                          struct Elemento * left, * right;  } Nodo;
typedef Nodo * Tree;
```

- La seguente funzione inserisce nell'albero *t* tutte le parole contenute nella lista *l*. L'indice deve contenere una sola volta le parole del testo, ordinate alfabeticamente secondo il criterio per cui in ogni nodo **n** dell'albero tutte le parole del sottoalbero destro precedono la parola di **n**, mentre quelle del sottoalbero sinistro la seguono.

```
Tree creaIndice( ListaParole l ) {
  Tree t = NULL;
  while( l != NULL ) {
    t = inserisciOrd( t, l->word );
    list = list->next;
  }
  return t;
}
```

- Si codifichi in C la funzione inserisciOrd, badando ad allocare i nodi per le parole non presenti nell'indice e aumentare il contatore delle occorrenze per le parole già presenti.

```
Tree inserisciOrd( Tree t, char * p ) {
  if( t == NULL ) {
    t = (Tree) malloc(sizeof(Nodo));
    t->left = t->right = NULL;
    t->occorrenze = 1;
    strcpy(t->parola, p);
  }
  else if ( strcmp(p, t->parola)==0 )
    t->occorrenze += 1;
  else if ( strcmp(p, t->parola) < 0 )
    t->left = inserisciOrd( t->left, p );
  else
    t->right = inserisciOrd( t->right, p );
  return t;
}
```

Esercizio (tde 18-2-2009)

- Si consideri la seguente definizione di un albero binario (binario=con due rami in ogni nodo):

  typedef struct EL { int dato;

  struct EL * left, * right; } node;

  typedef node * tree;

- Si scriva una funzione che prende in ingresso un albero binario e restituisce 1 se tutti i nodi godono delle proprietà di avere come discendenti a sinistra solo nodi con valori più piccoli e a destra solo nodi con valori più grandi, 0 altrimenti.

```
int max(int a,int b) {
      if(a>b)
            return a;
      else
            return b;
}

int max3(int a,int b,int c) {
      return max(a,max(b,c));
}

int min(int a,int b) {
      if(a<b)
            return a;
      else
            return b;
}

int min3(int a,int b,int c) {
      return min(a,min(b,c));
}
```

```
int cercaMax(tree t) {
  int s,d,m;
  if (t==NULL)
     return 0;//eseguita solo se da subito l'albero è vuoto
  if (t->left==NULL && t->right==NULL)
     return t->info;
  if(t->left==NULL)
     return max(t->info, cercaMax(t->right));
  if(t->right==NULL)
     return max(t->info, cercaMax(t->left));
  return max3(cercaMax(t->right),cercaMax(t->left),t->info);
}

int cercaMin(tree t) {
  int s,d,m;
  if (t==NULL)
     return 0;//eseguita solo se da subito l'albero è vuoto
  if (t->left==NULL && t->right==NULL)
     return t->info;
  if(t->left==NULL)
     return min(t->info, cercaMin(t->right));
  if(t->right==NULL)
     return min(t->info, cercaMin(t->left));
  return min3(cercaMin(t->right),cercaMin(t->left),t->info);
}
```

```
int f(tree t) {
    int max,min;
    if(t==NULL)
        return 1;
    if(t->left==NULL && t->right==NULL)
        return 1;

    if(t->left!=NULL) {
        max=cercaMax(t->left);
        if(t->dato<max)
            return 0;
    }
    if(t->right!=NULL) {
        min=cercaMin(t->right);
        if(t->dato>min)
            return 0;
    }

    return f(t->left) && f(t->right);
}
```

Esercizio (tde 9-6-2009)

- Si consideri la seguente definizione di un albero binario (binario=con due rami in ogni nodo):

  typedef struct EL { int dato;

  struct EL * left, * right; } node;

  typedef node * tree;

- Si definisce livello di un nodo la sua distanza dalla radice. Si scriva una funzione che prende in ingresso un albero binario e restituisce 1 se tutti i nodi di livello pari contengono un numero pari e tutti i nodi di livello dispari contengono un numero dispari.

```
int f(tree t){
    if(t==NULL)
        return 1;
    return f2(t,0);// o 1
}

int f2(tree t,int livello){
    if(t==NULL)
        return 1;
    if(livello%2!=t->dato%2)
        return 0;
    return f2(t->left,livello+1) && f2(t->right,livello+1);
}
```

Esercizio (tde 10-9-2010)

- Si consideri la seguente definizione di un albero binario:

  typedef struct EL { int dato;

  struct EL * left, * right; } node;

  typedef node * tree;
- Implementare una funzione che, ricevuti in ingresso due alberi binari TA e TB, restituisce 1 se la somma di tutti i nodi foglia di TA è uguale al valore di uno dei nodi di TB *oppure* se la somma di tutti i nodi foglia di TB è uguale al valore di uno dei nodi di TA.

```
int somma(Tree t){
    if(t==NULL)
        return 0;
    if(t->left==NULL && t->right==NULL)
        return t->info;
    return somma(t->left)+somma(t->right);
}

int cerca(Tree t,int val) {
    if(t==NULL)
        return 0;
    if(t->info==val)
        return 1;
    return cerca(t->left,val) ||  cerca(t->right,val);
}

int f(tree ta,tree tb) {
    return cerca(ta,somma(tb)) || cerca(tb,somma(ta));
}
```

Esercizio (tde 8-2-2011)

- Si consideri la seguente definizione di un albero binario:

```
typedef struct ET { char * dato;
                    struct ET * left, * right; } treeNode;
typedef treeNode * tree;
```

e la seguente definizione di lista:

```
typedef struct EL { char * dato;
                    struct EL * next; } listNode;
typedef listNode * list;
```

- Si assuma che gli elementi della lista siano ordinati alfabeticamente.

1. Si codifichi in C la seguente funzione:
   int contains (tree t, char * word)
   che restituisce 1 se l'albero contiene la parola data, 0 altrimenti
2. Si codifichi in C la seguente funzione:
   int isContained (tree t, list l)
   che restituisce 1 se almeno una parola in ogni livello dell'albero è contenuta nella lista

1.

```
int contains(Tree t,char * val) {
    if(t==NULL)
        return 0;
    if(strcmp(t->dato,val)==0)
        return 1;
    return contains(t->left,val) ||  contains(t->right,val);
}
```

2.

```
int containsInLista(Lista lis,char * val) {
    if(lis==NULL)
        return 0;
    if(strcmp(lis->dato,val)==0)
        return 1;
    return containsInLista(lis->next,val);
}

int depth ( tree t ) {
 int D, S;
 if (t == NULL)
   return 0;
 S = depth( t->left );
 D = depth( t->right );
 if ( S > D )
    return S + 1;
 else
    return D + 1;
}
```

```
void f1(tree t,Lista lis,int livello,int v[]) {
    if(T==NULL)
        return;
    if(containsInLista(lis,t->dato))
        v[livello]=1;
    f1(t->left,lis,livello+1,v);
    f1(t->right,lis,livello+1,v);
}

int f (tree t,Lista lis) {
    int * v,p,i=0,ris=1;
    p=depth(t);
    v=malloc(sizeof(int)*p);
    for(i=0;i<p;i++)
        v[i]=0;
    f1(t, lis,0,v);
    for(p=p-1;p>=0;p--)
        if(v[p]==0)
            ris=0;
    free(v);
    return ris;
```

2.

```
int containsInLista(Lista lis,char * val) {
    if(lis==NULL)
        return 0;
    if(strcmp(lis->dato,val)==0)
        return 1;
    return containsInLista(lis->next,val);
}

int inLivelloCeParola(tree t,int p,Lista lis,int liv){
    if(t==NULL)
        return 0;
    if(liv==p){
        if(containsInLista(lis,t->dato))
            return 1;
        else
            return 0;
    }
    return inLivelloCeParola(t->left,p,lix,liv+1) || inLivelloCeParola(t->right,p,lix,liv+1);
}
```

```
int f (tree t,Lista lis) {
        int * v,p,i=0;
        p=depth(t);

        for(p=p-1;p>=0;p--){

        if(!inLivelloCeParola(t,p,lis,0))

                        return 0;
        }

        return 1;
}
```

Esercizio (tde 6-7-2012)

- Si consideri la seguente definizione di un albero ternario:

```
typedef struct ET { int dato;
                    struct ET * left, * center, * right; } treeNode;
typedef treeNode * tree;
```

- Si codifichi in C la seguente funzione:

```
int cerca(tree t, int totale)
```

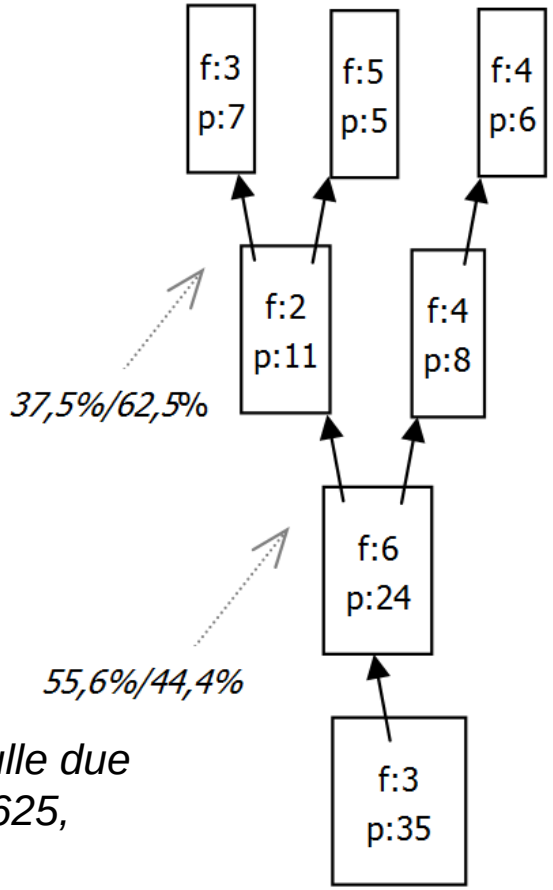
che restituisce 1 se esistono almeno due cammini distinti dalla radice alle foglie in cui la somma dei numeri contenuti nei nodi è uguale a totale. Due cammini sono distinti se differiscono anche solo per un nodo.

```
int cerca(Tree t, int totale) {
	if(t==null)
		return 0;
	if(cercaAux(t, totale)>=2)
		return 1;
	else
		return 0;
}

int cercaAux(Tree t, int totale) {
	if(t==null)
		return 0;
	if(t->left==null && t->center==null && t->right==null) {
		if(totale==t->info)
			return 1;
		else
			return 0;
	} else
		return cercaAux(t->left, totale - t->info) + cercaAux(t->center, totale - t->info)
				+ cercaAux(t->right, totale - t->info);
}
```

Esercizio (tdeB 7-2-2012)

- Un informatico ha lasciato in giardino, con la radice conficcata nel terreno, vari alberi binari del tipo definito sopra. Su rami e rametti si sono posati fiocchi di neve (in numero indicato da fiocchi), ma ogni ramo può sopportare senza spezzarsi al più un peso totale pari al valore di portata (espresso in fiocchi di neve). Ovviamente ogni ramo regge non solo i fiocchi direttamente posati su di esso, ma anche il peso di quelli posati sulle sue diramazioni (del resto i rami sono tanto più spessi quanto più sono vicini alla radice).
- Si noti che nell'albero in figura l'aggiunta di un solo fiocco su uno qualsiasi dei rami (ad eccezione della radice) farebbe spezzare l'albero in qualche punto.
- Si codifichi in C la funzione resiste(...) che restituisce **1** se l'albero regge, **0** se la neve lo spezza in un qualsiasi punto
- Si codifichi in C la funzione piegato(...) che restituisce **1** se in ogni diramazione effettiva il peso è ripartito con uno sbilanciamento massimo del 40%/60%, **0** se c'è anche solo una diramazione che non rispetta la proporzione.

```
typedef struct TB {
    long int fiocchi, portata;
    struct TB *left, *right;
} Ramo;

typedef Ramo * Tree;
```



Le percentuali indicano la ripartizione dei pesi sulle due uniche diramazioni effettive (3/8=0.375, 5/8=0.625, 10/18=0.5555.., 8/18=0.4444..).

```
int sommaF(Tree T) {
    if(T==NULL)
        return 0;
    return T->fiocchi+ sommaF(T->left)+sommaF(T->right);
}

int resiste(Tree t) {
    if(T==NULL)
        return 1;
    if(T->portata < sommaF(T))
        return 0;
    return resiste(T->left) && resiste(T->right);
}
```

```
int equilibrato(int a,int b){
    if(a==0 && b==0)
        return 1;
    if(b==0)
        return 0;
    if(((float)a/b<3.0/2) && ((float)a/b>2.0/3))
        return 1;
    return 0;
}
int f(Tree T) {
    if(T==NULL)
        return 1;
    if(!equilibrato(sommaF(T->left),sommaF(T->right)))
        return 0;
    return f(t->left)&& f(t->right);
}
```

Esercizio (tde 7-9-2009)

- Si consideri la seguente definizione di un albero binario (binario=con due rami in ogni nodo):

  typedef struct EL { int dato;

  struct EL * left, * right; } node;

  typedef node * tree;

- Un albero si definisce **lionese** se tutti i figli destri di nodi sono pari e tutti i figli sinistri sono dispari. La radice dell'albero può contenere qualsiasi valore in quanto non è né figlio destro né sinistro. Si scriva una funzione che riceve in ingresso un albero e restituisce 1 se l'albero è lionese, 0 altrimenti. Si suggerisce l'uso di funzioni ausiliarie.

```
int f(Tree T) {
   if(T==NULL)
        return 1;
   if(T->left==NULL && t->right==NULL)
        return 1;
   if(T->left!=NULL && T->left->dato %2==0)
        return 0;
   if(T->right!=NULL && T->right->dato %2==1)
        return 0;
   return f(t->left)&& f(t->right);
}
```

Esercizio (tde 9-2-2010)

- Si consideri la seguente definizione di un albero binario (ovvero con due rami in ogni nodo):

  typedef struct EL { int dato;

  struct EL * left, * right; } node;

  typedef node * tree;
- Codificare una funzione che riceve in input due alberi e restituisce 1 se tutti gli interi contenuti in un albero sono anche contenuti nell'altro e viceversa (anche più di una volta e in posizioni qualsiasi), 0 altrimenti.
- **Si faccia attenzione che gli alberi possono avere numeri duplicati e possono avere un numero complessivo di nodi differente.**

```
int cerca(tree t, int n) {
   if (t == NULL) {
      return 0;
   }
   if (t->dato == n) {
      return 1;
   }
   return cerca(t->left, n) || cerca(t->right, n);
}

int aux(tree orig, tree copia) {
   if (orig == NULL) {
      return 1;
   }
   if (cerca(copia, orig->dato) == 0) {
      return 0;
   }
   else {
      return aux(orig->left, copia) && aux(orig->right, copia);
   }
}

int f(tree a, tree b) {
   if (a == NULL || b == NULL) {
      return 0;
   }
   return aux(a, b) && aux(b, a);
}
```

Si consideri la seguente definizione di un albero binario:

```
typedef struct ET { int * dato;
                    struct ET * left;
                    struct ET * right; } treeNode;
typedef treeNode * tree;
```

Definiamo grado di un livello di un albero la somma degli elementi appartenenti al livello stesso.

Una coppia alberi t1 e t2 si dice k-similare se t1 ha esistono almeno k livelli in t1 e in t2 di pari grado.

Nota: non è necessario che i livelli di pari grado siano lo stesso livello nei due alberi.

Si supponga, per semplicità, che all'interno dello stesso albero non esistano due livelli dello stesso grado.

Si codifichi in C la seguente funzione:

```
int ksimilare(tree t1, tree t2, int k)
```

che restituisce 1 se t1 e t2 sono k-similari, 0 altrimenti.

```
void s(Tree T,int * v, int liv) {
    if(T==NULL)
        return;
    v[liv]+=T->dato;
    s(T->left,v,liv+1);
    s(T->right,v,liv+1);
}
int f(Tree T1,Tree T2,int k){
    int i,j,cont=0;d1,d2,* v1,* v2;
    d1=depth(T1); d2=depth(T2);
    v1=malloc(sizeof(int)*d1);
    v2=malloc(sizeof(int)*d2);
```

Esercizio (tde 25-2-2010)

- Si consideri la seguente definizione di un albero binario:

  typedef struct EL { int dato;

  struct EL * left, EL * right; } node;

  typedef node * tree;
- Implementare una funzione che, ricevuto in ingresso un albero binario, restituisce 1 se tutti i nodi sono o nodi foglia o godono della proprietà di avere tra i loro discendenti almeno un nodo divisibile per il valore che contengono. È possibile implementare funzioni di supporto.

```
int divisibile(int n, tree t) {
   if (t == NULL) {
      return 0;
   }
   if (t->dato % n == 0) {
      return 1;
   }
   return divisibile(n, t->left) || divisibile(n, t->right);
}

int f(tree t) {
   if (t == NULL) {
      return 1;
   }
    if (t->left == NULL && t->right == NULL) {
      return 1;
   }
   if(t->dato==0)
      return 0;
   return (divisibile(t->dato, t->left) || divisibile(t->dato, t->right)) && f(t->left) && f(t->right);
}
```

Esercizio (tde 9-7-2010)

- Si consideri la seguente definizione di un albero binario:

  typedef struct EL { int dato;
  struct EL * left, * right; } node;

  typedef node * tree;
- Implementare una funzione che, ricevuti in ingresso due alberi binari TA e TB, restituisce 1 se tutti i nodi foglia di TA sono nodi interni di TB e se tutti i nodi foglia di TB sono nodi interni di TA.
- È possibile implementare funzioni di supporto.

Esercizio (tde 24-2-2011)

- Si consideri la seguente definizione di un albero binario:

  typedef struct ET { int * dato;

  struct ET * left, * right; } treeNode;

  typedef treeNode * tree;
- Definiamo grado di un livello di un albero la somma degli elementi appartenenti al livello stesso.
- Una coppia alberi t1 e t2 si dice k-similare se t1 ha esistono almeno k livelli in t1 e in t2 di pari grado.
- Nota: non è necessario che i livelli di pari grado siano lo stesso livello nei due alberi.
- Si supponga, per semplicità, che all'interno dello stesso albero non esistano due livelli dello stesso grado.
- Si codifichi in C la seguente funzione:

  int ksimilare(tree t1, tree t2, int k)

  che restituisce 1 se t1 e t2 sono k-similari, 0 altrimenti.

Esercizio (tde 8-7-2011)

- Si consideri la seguente definizione di un albero binario:

  typedef struct ET { int dato;

  struct ET * left, * right; } treeNode;

  typedef treeNode * tree;
- Definiamo un albero “oxfordiano” se per ogni percorso possibile dalla radice alle foglie la somma degli elementi incontrati nel percorso è divisibile per tre. Si codifichi in C la seguente funzione:

  int oxfordiano(tree t)

  che restituisce 1 se t è “oxfordiano”, 0 altrimenti.

```
int aux(tree t, somma) {
    if (t == NULL) {
        if (somma % 3 == 0)
            return 1;
        else
            return 0;
    } else
        return aux(t->left, somma + t->dato) && aux(t->right, somma + t->dato);
}

int oxfordiano(tree t) {
    if (t == NULL)
        return 0;
    return aux(t, 0);
}
```

Esercizio (tde 12-9-2011)

- Si consideri la seguente definizione di un albero binario:

  typedef struct ET { char dato;
  struct ET * left, * right; } treeNode;

  typedef treeNode * tree;

  e la seguente definizione di lista

  typedef struct ch { char c; struct ch * next; } carattere;

  typedef carattere * parola;
- Si codifichi in C la funzione

  int f(tree t, parola p)

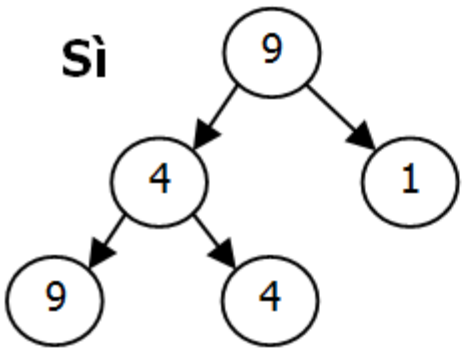
  che restituisce 1 se per in almeno un percorso dalla radice alle foglie la sequenza di caratteri incontrata (ordinata dalla radice alla foglia) è esattamente la parola contenuta nella lista p (si suggerisce l'uso di funzioni ausiliarie).

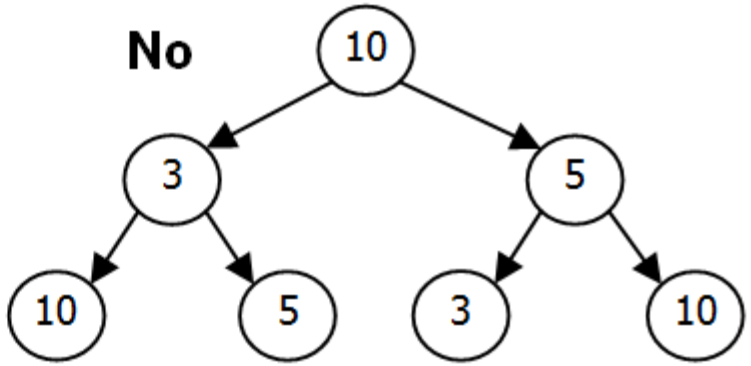
Esercizio (tdeB 9-2-2011)

typedef struct TN { int valore; struct TN * left, * right; } Node;

typedef Node * Tree;

- In un albero binario di interi, rappresentato secondo la precedente definizione, si definiscono *interni* tutti i nodi che non sono nodi foglia. Si definisca e codifichi in C la funzione **… ridondato(…)** che verifica (restituendo 1 in caso affermativo, 0 altrimenti) se un albero gode della seguente proprietà: un albero si dice *ridondato* se ogni valore *vi* presente su un nodo interno *ni* è anche presente in un nodo foglia *nf* dell'albero che ha *ni* come radice.
- Si noti che il secondo albero nell'esempio sottostante non è ridondato, anche se tutti i valori interni (10,3,5) sono presenti anche su qualche foglia dell'albero.





Esercizio (tde 10-9-2012)

- Si consideri la seguente definizione di un albero ternario:

```
typedef struct ET { int dato;
                    struct ET * left, * center, * right; } treeNode;
typedef treeNode * tree;
```

- Si codifichi in C la seguente funzione:

```
int f(tree t)
```

che restituisce 1 se in tutti i cammini dalle foglie alla radice il valore contenuto nel campo “dato” di un nodo (nodo radice escluso) non è superiore a 2 più il valore contenuto nel campo “dato” del padre.

Esercizio (tde 5-2-2013)

- Si consideri la seguente definizione di un albero binario:

```
typedef struct EL { int dato;
                    struct EL * left, * right; } node;
typedef node * tree;
```

- Implementare una funzione che, ricevuti in ingresso due alberi binari TA e TB, restituisce 1 se TA contiene un in un nodo un valore uguale alla somma di tutti i nodi foglia di TB e se TB contiene un in un nodo un valore uguale alla somma di tutti i nodi interni di TA. È consigliato implementare funzioni di supporto.

```
int sommaF(tree T) {
    if(T==NULL) return 0;
    if(T->left==NULL && T->right==NULL)
        return T->dato;
    return sommaF(T->left) + sommaF(T->right);
}
int sommaI(tree T) {
    if(T==NULL) return 0;
    if(T->left==NULL && T->right==NULL)
        return 0;
    return T->dato+sommaF(T->left) + sommaF(T->right);
}
int esiste(tree T,int num){
    if(T==NULL) return 0;
    if(T->dato==num)
        return 1;
    return esiste(T->left,num) || esiste(T->right,num) ;
}
int f(tree TA,tree TB){
    return esiste(TA,sommaF(TB)) && esiste(TB,sommaI(TA));
}
```

Esercizio (tde 27-2-2013)

- Si consideri la seguente definizione di un albero binario

```
typedef struct EL { int dato;
                    struct EL * left, right; } node;
typedef node * tree;
```

- Si scriva una funzione che prende in ingresso un albero binario e restituisce 1 se è completo (cioè se tutti i nodi o sono foglie o hanno 2 figli e se tutte le foglie sono alla stessa profondità), 0 altrimenti.

```
int aux(tree t, counter) {
    int fromLeft;
    int fromRight;

    if (t->left == NULL && t->right == NULL) {
            return counter;
    } else if ((t->left == NULL && t->right != NULL) ||
           (t->left != NULL && t->right == NULL)) {
            return -1;
    } else {
            fromLeft = aux(t->left, counter + 1);
            fromRight = aux(t->right, counter + 1);
            if (fromLeft != fromRight) {
                    return -1;
            } else {
                    return fromLeft;
            }
    }
}
```

```
int f(tree t) {
    if (t == NULL) {
        return 0;
    }
    if (aux(t, 0) >= 0) {
        return 1;
    } else {
        return 0;
    }
}
```